组成原理实验课程第_四_次实报告

实验名称	ALU 模块实现			班级	李涛
学生姓名	李佳豪	学号	2111252	指导老师	董前琨
实验地点	A306		实验时间	2023.5.9	

1 实验目的

(1) 熟悉 MIPS 指令集中的运算指令，学会对这些指令进行归纳分类。
(2) 了解 MIPS 指令结构。
(3) 熟悉并掌握 ALU 的原理、功能和设计。
(4) 进一步加强运用 verilog 语言进行电路设计的能力。
(5) 为后续设计 cpu 的实验打下基础。

2 实验内容说明

2.1 学习 ALU 的工作原理, 使用 vivido 进行模拟实现。

- 根据 ALU 工作原理编写 Verilog 代码
- 实验箱进行尝试 12 个算数功能，并进行手动验证

2.2 对原始 ALU 代码进行修改

- 压缩 ALU 运算器的符号控制独热码至 4 位
- 增加三个自定义运算加入 ALU 中

3 实验原理图

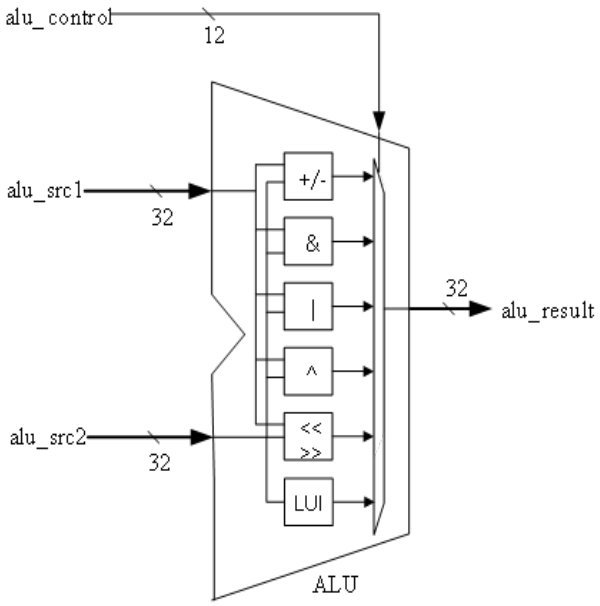

图 1: ALU 原理图

4 实验步骤

4.1 表格

操作码	操作类型	解析与例子
800	加	ABCE + 123 = ACFE
400	减	ABCE - 123 = AAAB
200	无符号置位	$12 < 15 = 1$
100	有符号置位	$A1110000 < 1 = 1$
080	与	DBC $\wedge$ ACBE = CBC
040	或非	$\neg$(DBC $\vee$ ACBE)=5421
020	或	DBC $\vee$ ACBE = ABDE
010	异或	$DBC \oplus$ACBE =
008	逻辑左	123 « 2 = 48C
004	逻辑右	123 » 2 = 48
002	算数右	213 » 2 = 84
001	高位加载	12345678 == 1234

操作码	操作类型	解析与例子
800	加	ABCE + 123= ACFE
400	减	ABCE - 123= AAAB
200	无符号置位	$12 < 15 = 1$
100	有符号置位	$A1110000 < 1 = 1$
080	与	$DBC \wedge ACBE = CBC$
040	或非	$\neg(DBC \vee ACBE) = 5421$
020	或	$DBC \vee ACBE = ABDE$
010	异或	$DBC \oplus ACBE =$
008	逻辑左	$123 \ll 2 = 48C$
004	逻辑右	$123 \gg 2 = 48$
002	算数右	$213 \gg 2 = 84$
001	高位加载	$12345678 == 1234$
D	算数左	$2 \ll 4 = 20$
E	有符号大于置为	$A > 0 = 1$
F	低位加载	$A9577655 == 7655$

4.2 上述表格 ALU 操作代码原理

- 加减法

  利用之前实验的 adder

- 置为操作

  使用 adder 计算两个操作数的差，若为有符号数，根据差的符号位和两个操作数的符号位，由此化简真值表得到结果; 对于 32 无符号位比较，在其最高位前填 0 作为 33 位正数比较。

- 逻辑操作

  按位与、或、非、异或等。

- 左、右移

  根据移动位数来按位左移和右移几位，注意算数左移和逻辑右移实际是一样的

- 高低位加载

  读取指定操作数的高、低 16 位的值，使用 veilog 的切片操作即可。

4.3 代码修改

1. 压缩控制信号至 4 位

```
module alu(
    input [3:0] alu_control,
    // input [11:0] alu_control, // ALU控制信号
    input [31:0] alu_src1,  // ALU操作数1,为补码
    input [31:0] alu_src2,  // ALU操作数2, 为补码
    output [31:0] alu_result // ALU结果
    );
```

2. 根据 4 位控制信号进行独热编码，且增加三个独热码

- 增加独热码

```
    wire alu_sla; //算数左移
    wire alu_llt ; // 大于置为
    wire alu_lli; // 低位加载
```

- 4 位控制信号转为独热码

```
    assign alu_add//0001
    = !alu_control[3]&&!alu_control[2]&&!alu_control[1]&&alu_control[0];
    assign alu_sub//0010
    = !alu_control[3]&&!alu_control[2]&&alu_control[1]&&!alu_control[0];
    assign alu_slt//0011
    = !alu_control[3]&&!alu_control[2]&&alu_control[1]&&alu_control[0];
    assign alu_sltu//0100
    = !alu_control[3]&&alu_control[2]&&!alu_control[1]&&!alu_control[0];
    assign alu_and//0101
    = !alu_control[3]&&alu_control[2]&&!alu_control[1]&&alu_control[0];
    assign alu_nor//0110
    = !alu_control[3]&&alu_control[2]&&alu_control[1]&&!alu_control[0];
    assign alu_or//0111
    = !alu_control[3]&&alu_control[2]&&alu_control[1]&&alu_control[0];
    assign alu_xor//1000
    =alu_control[3]&&!alu_control[2]&&!alu_control[1]&&!alu_control[0];
    assign alu_sll//1001
    =alu_control[3]&&!alu_control[2]&&!alu_control[1]&&alu_control[0];
    assign alu_srl//1010
        =alu_control[3]&&!alu_control[2]&&alu_control[1]&&!alu_control[0];
    assign alu_sra//1011
        =alu_control[3]&&!alu_control[2]&&alu_control[1]&&alu_control[0];
```

```
21    assign alu_lui//1100
          =alu_control[3]&&alu_control[2]&&!alu_control[1]&&!alu_control[0];
22    assign alu_sla//1101
          =alu_control[3]&&alu_control[2]&&!alu_control[1]&&alu_control[0];
23    assign alu_llt//1110
          =alu_control[3]&&alu_control[2]&&alu_control[1]&&!alu_control[0];
24    assign alu_lli//1111
          =alu_control[3]&&alu_control[2]&&alu_control[1]&&alu_control[0];
```

3. 增加的三个操作代码，代码解析见上节

- 算术左移

```
1       //算数左移
2     wire [31:0] sla_step1;
3     wire [31:0] sla_step2;
4     assign sla_step1 = {32{shf_1_0 == 2'b00}} & alu_src2         //
          若shf[1:0]="00",不移位
5                      | {32{shf_1_0 == 2'b01}} & {alu_src2[30:0], 1'd0}
                           // 若shf[1:0]="01",左移1位
6                      | {32{shf_1_0 == 2'b10}} & {alu_src2[29:0], 2'd0}
                           // 若shf[1:0]="10",左移2位
7                      | {32{shf_1_0 == 2'b11}} & {alu_src2[28:0], 3'd0};
                           // 若shf[1:0]="11",左移3位
8     assign sla_step2 = {32{shf_3_2 == 2'b00}} & sla_step1        //
          若shf[3:2]="00",不移位
9                      | {32{shf_3_2 == 2'b01}} & {sla_step1[27:0], 4'd0}
                           // 若shf[3:2]="01",第一次移位结果左移4位
10                     | {32{shf_3_2 == 2'b10}} & {sla_step1[23:0], 8'd0}
                           // 若shf[3:2]="10",第一次移位结果左移8位
11                     | {32{shf_3_2 == 2'b11}} & {sla_step1[19:0],
                           12'd0}; //
                           若shf[3:2]="11",第一次移位结果左移12位
12    assign sla_result = shf[4] ? {sla_step2[15:0], 16'd0} : sla_step2;
          // 若shf[4]="1",第二次移位结果左移16位
```

- 有符号大于置位

  根据有符号小于置为可以知道，其结果取反为有符号大于等于置位，此时单独考虑等于不置位的情况，即写作如下形式，避免了再次求真值表化简

```
1     assign llt_result[31:1] = 31'd0;
2     assign llt_result[0] = (add_sub_result != 0) & (~(slt_result[0]));
```

- 低位加载

```
        assign lli_result = {16'd0,alu_src2[15:0]}; ///低位加载
```

4. display 模块修改 alu_control 为压缩后的 4 位

```
    reg  [3:0] alu_control;
```

5 实验结果分析

5.1 验证算数左移，操作码为 D，实际和逻辑左移结果相同

2 左移 4 位，即扩大 16 倍, 结果位 32，16 进制为 20。

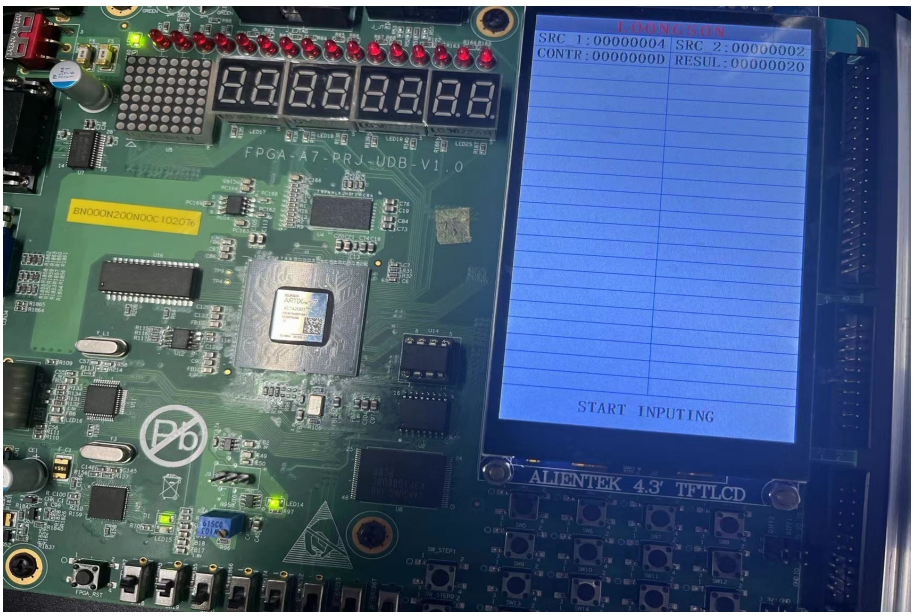

图 2: 2 逻辑左移 4 位

5.2 验证大于置为，操作码为 E

无符号数组 A 大于 0，则大于置位 1

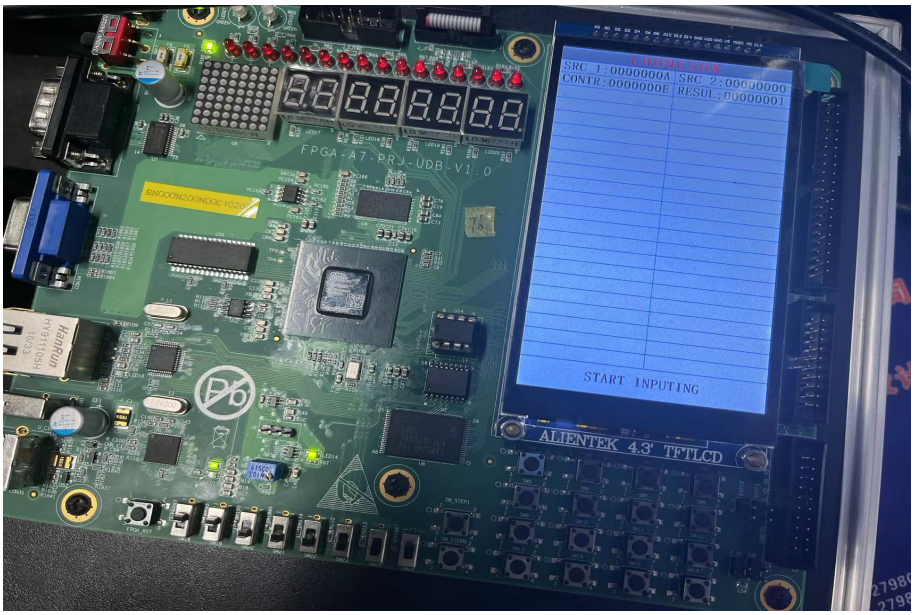

图 3: A 大于 0 置为 1

5.3 验证低位加载，操作码为 F

低位加载 A9577655，显示 7655

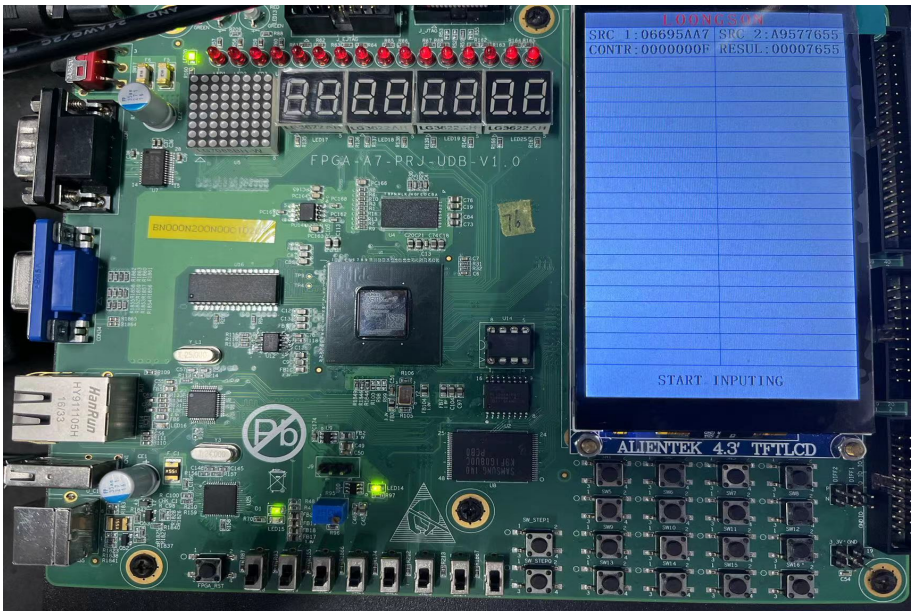

图 4: 低位加载 A9577655

6 总结感想

1. 学习了 ALU 应对不同指令时内部的工作原理，对于不同的指令会产生对于的控制码，而 ALU 会产生所有操作类型的结果，但最后只输出控制码对应的结果。

2. 为数据通路和 CPU 设计的学习有了辅助和铺垫。